The vi Editor: Last Look

The discussion of the vi editor began in Chapter 4 and continues in this chapter. Chapter 6 describes more of the vi editor's power and flexibility and introduces more advanced commands. It explains the commands' scopes and their usage in combination with the other commands, discusses the vi editor's manipulation of temporary buffers, and shows some of the ways the vi editor can be customized to your needs. By the end of this chapter, you will be well equipped to use vi to do your editing jobs.

In This Chapter

6.1 MORE ABOUT THE vi EDITOR

The vi editor is part of the *ex family* of editors. vi is the screen-oriented part of the ex editor, and it is possible to switch between the vi and ex editors. In fact, vi commands that start with **:** are ex editor commands. While in vi's command mode, pressing **:** displays the colon prompt at the bottom of the screen and causes vi to wait for your command. When you complete your command line by pressing **[Return]**, ex executes the command and, upon completion, returns control to vi.

To change to the ex editor, type **:Q** and press **[Return]**. If you have intentionally or accidentally changed to the ex editor, type **vi** to get back to the vi editor, or type **q** to exit the ex editor and return to the shell prompt.

6.1.1 Invoking the vi Editor

In Chapter 4, you learned how to start vi, how to save a file, and how to quit vi. Expanding on those commands, let's explore other ways the vi editor can be invoked and ended.

You can start the vi editor without providing a filename. In this case, you use the write (**:w**) or write and quit (**:wq**) command to name your file.

The following command sequences show you how to invoke vi without a filename and later to name and save your file:

- Type **vi** and press **[Return]** to invoke the vi editor without a filename.
- Type **:w myfirst** and press **[Return]** to save the contents of the temporary buffer into the `myfirst` file and stay in the vi editor.
- Type **:wq myfirst** and press **[Return]** to save the contents of the temporary buffer into the `myfirst` file and quit the vi editor.

If the file you are currently editing does not have a name and you type **:w** or **:wq** without giving a filename, vi displays the following message:

```
No current filename
```

The vi editor normally prevents overwriting an existing file. Thus if you type **:w myfirst** and press **[Return]**, and `myfirst` already exists, then vi warns you by showing the following message:

```
"myfirst" File exists - use ":w! to overwrite"
```

If you want to overwrite an existing file, use the **:w!** command.

The write command (**:w**) is also useful if you want to save your file or part of it under another name and keep the original file intact. The following command sequences show ways you can name a file or change the name of the current editing file:

- Type **vi myfirst** and press **[Return]** to invoke vi and copy the `myfirst` file into the temporary buffer.

- Type **:w yourfirst** and press **[Return]** to save the contents of the temporary buffer (`myfirst`) into the `yourfirst` file. Your current editing file remains `myfirst`. The display shows this message:

```
"yourfirst" [New file] 3 lines, 103 characters
```

- Type **:wq yourfirst** and press **[Return]** to save the temporary buffer into the `yourfirst` file and quit the vi editor. The original `myfirst` file remains intact.

6.1.2 Using the vi Invocation Options

The vi editor provides flexibility from the very start. You can invoke vi with certain invocation options that you type as part of the command line.

The Read Only Option The **-R** (for read only) option makes a file a read only file and allows you to step through the contents of the file without making accidental changes. To use this option with the `myfirst` file, type **vi -R myfirst** and press **[Return]**. The vi editor shows the following message on the bottom line of the screen:

```
"myfirst" [Read Only] 3 lines, 106 characters
```

If you try to save the read only file using the **:w** or the **:wq** command, vi displays a message indicating this is a read only file:

```
"myfirst" File is read only
```

Use the **-R** option with the `myfirst` file.

```
$ vi -R myfirst [Return] . . . . .
```
Use the read only option.

The vi editor opens the `myfirst` file and shows a message that indicates this is a read only file.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
~
"myfirst" [readonly] 3L, 116C
```

If you try to save a read only file, using the **:w** or **:wq** command, vi displays a message indicating this is a read only file.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
~
:wq
'readonly' option is set (use ! to override)
```

If you want to save a read-only file, you must force the write operation by using the **!** option. In our example, the command **:wq!** overrides the read-only option and writes the file.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
:wq!
"myfirst" 3L, 116C written
$
```

You can override read-only files that are your files or if you have access to them. Most of the system files are read-only files, however, and a normal user cannot change this option for them.

Quitting a Read-Only File Generally, you open a file as read only file just to look at it or read it, and your intention is not to edit and save it. To quit a read-only file, you type **:q** [Return] and to force it to quit, you type **:q! [Return]**.

Viewing Files You can use **view** to open the vi editor in read-only mode. **view** is a version of vi that always starts in read only mode. That is, you will be protected from writing over the files. You also can start vi in read-only mode by using the **-R** option, as was explained in the previous section.

The following two command lines will both open `myfirst` in read-only mode:

```
$ view myfirst [Return]   . . . . . Use view to read a file.
$ vi -R myfirst [Return]  . . . . . Use vi with read only option to
                                    read a file.
```

The Command Option The **-c** (for command) option allows you to give specific vi commands as part of the command line. This option is useful to position the cursor or search for a pattern in a file before you begin editing. To use this option with the `myfirst` file, type **vi -c /most myfirst** and press **[Return]**. You are giving a search command (**/most**) as part of the command line. The vi editor copies the `myfirst` file into the temporary work buffer and places the cursor on the line with first occurrence of the word *most.*

Use the **-c** option with the `myfirst` file:

```
$ vi -c /most myfirst [Return] . .  Use the –c command option.
```

-c indicates the command option is used.

/most indicates you are giving a search command as part of the command line.

The vi editor opens the `myfirst` file (copies the `myfirst` file into a temporary work buffer) and places the cursor on line two that happens to have the first occurrence of the word *most*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
"myfirst" 3L, 116C
```

6.1.3 Editing Multiple Files

You can start vi and give it a list of filenames instead of one filename. Then when you finish editing one file, you can start editing the next file without reinvoking the vi editor. You press **:n** (for next) to invoke the next editing file. When you issue the **:n** command, vi replaces the contents of the working buffer with text from the next file. However, if you have modified the current text, vi displays a message like the following:

```
No write since last change (:next ! overrides)
```

You can use the **n!** command to override this protection. In this case, the changes you have made in your current editing file are lost.

To see a list of the filenames, you use the **:ar** command. vi responds by showing a list of the filenames and also indicates the name of the file being edited.

We are using `file1` and `file2` for this terminal session, and we need a file called `yourfirst` for next terminal session. You can create these files quickly using the vi editor. The following screen shows the content of each file.

`$` **`vi file1 [Return]`** `. . . . .` Create the `file1` file.

```
file1
This is file 1.
~
:wq
"file1" 2L, 22C written
$
```

`$` **`vi file2 [Return]`** `. . . . .` Create the `file2` file.

```
file2
This is file 2.
~
:wq
"file2" 2L, 22C written
$
```

`$` **`vi yourfirst [Return]`** `. . .` Create the `yourfirst` file.

```
yourfirst
This is "yourfirst" file.
~
:wq
"yourfirst" 2L, 36C written
$
```

The following command sequences show examples of specifying more than one filename in a command line:

- Type **vi file1 file2** and press **[Return]** to invoke vi with two editing files, `file1` and `file2`; vi responds:

```
2 files to edit
"file1" 2 lines, 22 characters
```

```
file1
This is file 1.
~
~
"file1" 2L, 22C
```

- Type **:w** and press **[Return]** to save `file1`.
- Type **:ar** and press **[Return]** to display the names of the files. vi indicates the filename of the current file by enclosing it in brackets:

```
[file1] file2
```

```
file1
This is file 1.
~
~
:ar
[file1] file2
```

- Type **:n** and press **[Return]** to start `file2` for editing; vi responds:

```
"file2" 2 lines, 22 characters
```

```
file2
This is file 2.
~
~
:n
"file2" 2L, 22C
```

❑ Type **:ar** and press **[Return]** to display the names of the files. This time the current file is `file2` and vi indicates that by enclosing the filename in brackets.

```
file1
This is file 1.
~
~
:ar
file1 [file2]
```

1. *If you type* **:n** *and there are no more files on the command line to open, the following message is displayed:*

   ```
   No more files to edit
   ```

2. *You can use* **:n** *to open the next file, but there is no way to go back and open the previous file.*

3. *If you type* **:n** *before saving your changes in the current file, vi displays the following message:*

   ```
   No write since last change (:Next! Overrides)
   ```

   In this case you can save your file (**:w**) *or type* **:n!** *and press* **[Return]** *to edit the next file without saving the current file.*

Editing Another File Another way to edit multiple files is to use the **:e** (for edit) command to switch to a new file. While in the vi editor, type **:e** followed by the name of the file and press **[Return]**. Usually, you save the current editing file before bringing in a new file and, unless you have made no changes, the vi editor warns you to write (save) the current file before switching to the next file.

Try the following command sequences to experiment with changing files:

❑ Type **vi** and press **[Return]** to invoke vi without specifying the filename.

```
~
~
:e myfirst
```

❑ Type **:e myfirst** and press **[Return]** to call in the `myfirst` file. Your current editing file is `myfirst`; vi shows the name and size of `myfirst`:

```
"myfirst" 3 lines, 103 characters
```

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
"myfirst" 3L, 103C
```

Reading Another File The vi editor lets you read (import) a file into your current editing file. While in the vi editor command mode, type **:r** followed by the name of the file and press **[Return]**. The **:r** command places a copy of the specified file into the buffer after the cursor position. The specified file becomes part of your current editing file.

Use the following command sequence to import (read) another file into your current working buffer:

- Type **vi myfirst** and press **[Return]** to invoke the vi editor and edit `myfirst`.
- Type **:r yourfirst** and press **[Return]** to add the contents of `yourfirst` to the current editing file. vi shows the name and size of the imported file:

```
"Yourfirst" 2 lines, 36 characters
```

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
:r yourfirst
```

Notice that the content of the `yourfirst` file is added right after the current line. These two lines are highlighted for your attention. We can control where a file is inserted by placing the cursor on the appropriate line. For example, you place the cursor on the last line if you want the imported file to be appended to the end of the file.

```
The vi history
yourfirst
This is "yourfirst" file.
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
"yourfirst" 2L, 36C
```

If you save this file (for example, using **:wq**), then the content of the `myfirst` file will be as is shown in this last screen example, with the two lines inserted after line one.

The **:r** *command adds a copy of the specified file to your current editing file, and the specified file remains intact.*

Writing to Another File The vi editor lets you write (save) a part of your current editing file into another file. You indicate the range of the lines you intend to save and use the **:w** command to write them. For example, if you want to save the text from lines 5 to 100 into a file called `temp`, type **:5,100 w temp** and press **[Return]**. vi saves lines 5 through 100 in a file called `temp` and shows a message similar to the following:

```
"temp" [new file] 96 lines, 670 characters
```

If the filename already exists, vi displays an error message. Then you can provide a new filename or use the **:w!** command to overwrite the existing file.

Save the first two lines in `myfirst` in a file called `temp`.

- Type **vi myfirst** and press **[Return]**. The vi editor is invoked with `myfirst` file.
- Type **:1,2 w temp** and press **[Return]**. The first two lines are saved in `temp`.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
~
~
:1,2 w temp
"temp" [New] 2L, 77C written
```

If the filename `temp` already exists, then you get a message such as

```
"temp" Use "w!" to write partial buffer
```

In this case, you type the **:w!** command to overwrite the existing file, or you can give it another filename.

- Type **:1,2 w! temp** and press **[Return]**. The first 2 lines are saved in `temp`. If `temp` already exists, it will be overwritten.
- Type **:1,2 w xyz** and press **[Return]**. The first two lines are saved in `xyz`, a new filename.
- Type **:e temp** and press **[Return]** to check the content of the `temp` file.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the
~
~
:e temp
"temp" 2L, 77C
```

6.2 REARRANGING TEXT

Deleting, copying, moving, and changing text are collectively referred to as *cut-and-paste operations*. Table 6.1 summarizes the operators or command keys that are used in combination to do cut-and-paste operations in a file. All of the commands are applicable when vi is in the command mode. With the exception of the change command, the vi editor remains in the command mode after completion of the command. The change command places the vi editor in text input mode, which means that you must press **[Esc]** to return vi to the command mode.

Table 6.1
The vi Editor Cut-and-Paste Keys

Key	Operation
d	Deletes a specified portion of the text and stores it in a temporary buffer. This buffer can be accessed by using the put operator (**p** or **P**).
y	Copies (Yanks) a specified portion of the text into a temporary buffer. This buffer can be accessed by using the put operator (**p** or **P**).
P	Places the contents of a specified buffer above the cursor position.
p	Places the contents of a specified buffer below the cursor position.
c	Deletes text and places vi in the text input mode. This is a combination of the delete and insert commands.

Assuming that you have the `myfirst` file on the screen and the cursor on *s,* the following examples show the cut-and-paste applications.

6.2.1 Moving Lines: dd and p or P

Using the delete and the put operators, you can move text from one part of a file to another:

- Press **dd**. vi deletes the current line, saves a copy of it in the temporary buffer, and moves the cursor to the *U.*
- Press **p**. vi places the deleted line below the current line.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
by most of the UNIX operating systems.
```

```
The vi history
supported by most of the UNIX operating systems.
The vi editor is an interactive text editor that is
```

- Use the cursor movement keys and place the cursor on any character on the first line.
- Press **P**. vi places the deleted line above the current line.

```
The vi editor is an interactive text editor that is
The vi history
supported by most of the UNIX operating systems.
The vi editor is an interactive text editor that is
```

The deleted text remains in the temporary buffer, and you can move a copy of it to different places in the file.

6.2.2 Copying Lines: yy and p or P

Using the *yank* and the *put* operators, you can copy text from one part of the file to another:

- Press **yy**. vi copies the current line into a temporary buffer.
- Use the cursor movement keys to place the cursor on the first line.
- Press **p**. vi copies the contents of the temporary buffer below the current line.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

- Use the cursor movement keys to move the cursor to the last line.
- Press **P**. vi copies the contents of the temporary buffer above the current line.

```
The vi history
The vi editor is an interactive text editor that is
The vi editor is an interactive text editor that is
The vi editor is an text editor which is
supported by most of the UNIX operating systems.
```

The copied text remains in the temporary buffer until the next delete or copy operation. You can copy the contents of this buffer to anywhere in the file and as many times as you wish.

6.3 SCOPE OF THE vi OPERATORS

In Chapter 4, you learned the basic vi commands; however, many vi commands operate on a block of text. A block of text can be a character, a word, a line, a sentence, or some other specified collection of characters. Using the vi commands in combination with the scope keys gives you more control over your editing task. The format for these types of commands can be represented this way:

```
command = operator + scope
```

There is no specific scope key to indicate the entire line. In order to indicate the entire line as the scope of a command, you press the operator key twice. For example, we saw that **dd** deletes a line and **yy** yanks (or copies) a line. Table 6.2 summarizes some of the common scope keys used in combination with other commands.

Table 6.2
Some of the vi Scope Keys

Scope	Operation
$	The scope is from the cursor position to the end of the current line.
0 (zero)	The scope is from just before the cursor position to the beginning of the current line.
e or **w**	The scope is from the cursor position to the end of the current word.
b	The scope is from the letter before the cursor backward to the beginning of the current word.

The following examples demonstrate the use of commands with the scope operators. To follow these examples, begin with the `myfirst` file on the screen. Using the delete, yank, and change operators on this file gives you a practical view of these commands.

6.3.1 Using the Delete Operator with Scope Keys

To delete text from the current cursor position to the end of the current line:

- Press **d$**. vi deletes text starting from the cursor position to the end of the current line and moves the cursor to the space after the word *by*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

You can use **u** *or* **U** *to undo your most recent text changes.*

```
The vi history
The vi editor is an interactive text editor that is
supported by_
```

To delete text starting from the cursor position to the beginning of the current line:

- Press **d0**. vi deletes the text and the cursor remains on the letter *m*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
most of the UNIX operating systems.
```

To delete a word after the cursor position:

- Press **dw**. vi deletes the word *most* and the space after it, and moves the cursor to the letter *o*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by of the operating systems.
```

To delete more than one word after the cursor position (for example, three words):

- Press **3dw**. vi deletes three words, *most*, *of*, and *the*, and the space after the, and moves the cursor to the space after the word *by*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi is an interactive text editor that is
supported by _UNIX operating systems.
```

To delete to the end of a word:

- Press **de**. vi deletes the word *most* and moves the cursor to the space before the letter *o*.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by of the operating systems.
```

To delete to the beginning of the previous word:

- Press **db**. vi deletes the word *by,* and the cursor remains on the space before the letter *o*.

```
The vi history
The vi editor is an interactive text editor that is
supported  of the operating systems.
```

6.3.2 Using the Yank Operator with Scope Keys

The yank operator can use the same scope keys as the delete operator. The **p** and **P** operators are used to place the yanked text in other places in the file. What portion of the text is yanked is controlled by using the scope keys.

To copy text from the current cursor position to the end of the current line, do the following:

- Press **y$**. vi copies the text starting from the cursor position to the end of the current line (is) to the temporary buffer, and the cursor remains on the letter *i*.
- Use the cursor movement keys to move the cursor to the end of the last line.
- Press **p**. vi copies the yanked text right after the cursor position.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.is
```

To copy text from the current cursor position to the beginning of the current line, do the following:

- Press **y0**. vi copies the text starting from the cursor position to the beginning of the current line (The vi), and the cursor remains on the letter *e.*
- Use the cursor movement keys to place the cursor at the end of the first line.
- Press **P**. vi copies the yanked text from the temporary buffer to right before the cursor position.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi historyThe vi
The vi editor is a text editor that is
supported by most of the UNIX operating systems.
```

6.3.3 Using the Change Operator with Scope Keys

The change operator, **c**, can use the same scope keys as the delete and yank operators. The difference in the **c** operator's function is that it changes vi from the command mode to the text input mode. After pressing **c**, you can enter text, beginning from the cursor position, and the text moves to the right. It wraps around when necessary to make room for the text you are entering. You return vi to command mode, as always, by pressing **[Esc]**. The change operator deletes the portion of the text indicated by the scope of the command and also places the vi editor in text input mode.

Some versions of the vi editor have a marker to mark the last character to be deleted. This marker is usually a dollar sign (**$**), and it overwrites the last character to be deleted.

The following example shows how to use the change text operator with the scope key to change a word:

- Press **cw**. vi places a marker at the end of the current word, overwrites the letter *t,* and changes to text input mode. The cursor remains on the letter *m,* the first character scheduled for change.
- Type **all** to change the word *most* to *all.*
- Press **[Esc]**. vi returns to command mode. Marker is removed.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by mos$ of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor that is
supported by all of the UNIX operating systems.
```

6.4 USING BUFFERS IN vi

The vi editor has several buffers used for temporary storage. The temporary buffer (work buffer) that holds a copy of your file was discussed in Chapter 4, and when you use the write command, the contents of this buffer are copied to a permanent file. There are two categories of temporary buffers, *numbered buffers* and *named buffers* (or *alphabetic buffers*), that you can use for storing your changes and later retrieving them.

6.4.1 The Numbered Buffers

The vi editor uses nine temporary buffers numbered from 1 to 9. Each time you delete or yank text, it is placed in these temporary buffers, and you can access any of the buffers by specifying the buffer number. Each new deletion or yanking of the text replaces the previous contents of these buffers. For example, when you give the **dd** command, vi stores the deleted line in buffer 1. When you use **dd** again to delete another line, vi bumps the old contents up one buffer, in this case to buffer 2, and then stores the new material in buffer 1. This means that buffer 1 always holds the most recently changed material. The contents of the numbered buffers change each time you issue a delete or yank command. After a few text changes, of course, you lose track of what is stored in each of the numbered buffers. But keep on reading; the best is yet to come!

The contents of the numbered buffers can be recovered by using the put operator, prefixed with the buffer number. For example, to recover the material from buffer 9, you type **"9p**. The command **"9p** means copy the contents of buffer 9 to where the cursor is. The format for specifying the buffer number can be presented as follows:

double quotation mark + *n* (where *n* is the buffer number from 1 to 9) + (**p** or **P**)

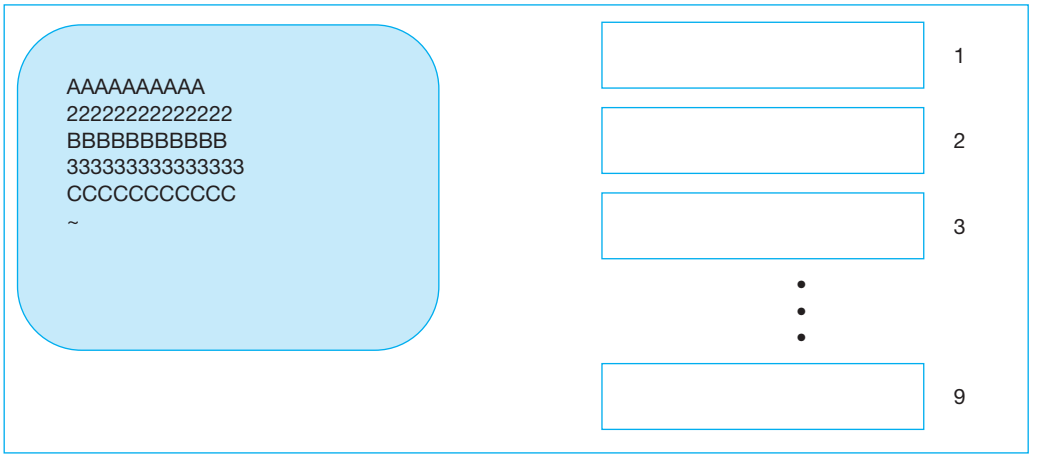
The temporary numbered buffers are depicted in Figures 6.1 through 6.5. The examples explain the vi editor's sequence of events when you change text in the file.

To practice the following terminal session, you need to create a file that contains text similar to the following lines of characters and numbers. Using vi, create a file called `buffer` under the `Chapter6` directory.

```
AAAAAAAAAA
222222222222222
BBBBBBBBBB
333333333333333
CCCCCCCCCC
```

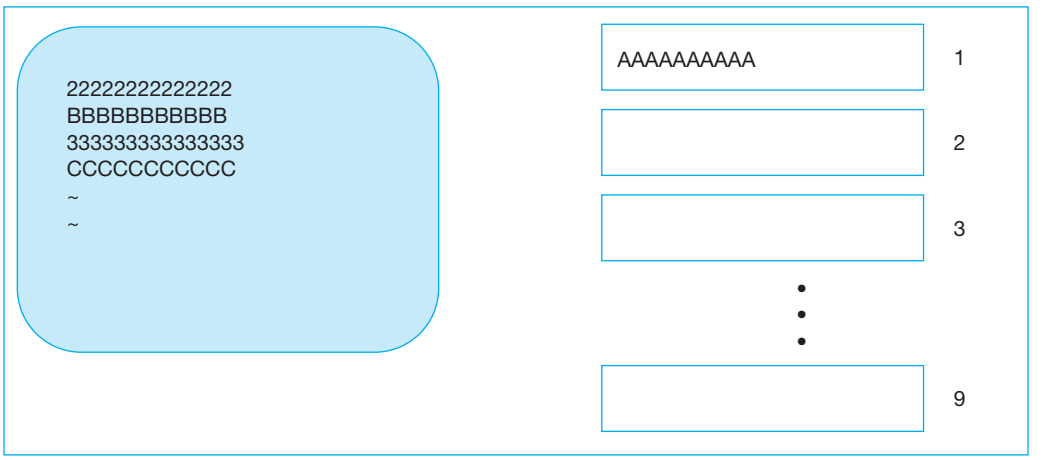
Assume that your screen looks like the screen in Figure 6.1, with five lines of text and the numbered buffers empty, because you have not yet done any editing operation.

Figure 6.1
The vi Editor's Nine Numbered Buffers



❑ Position the cursor on the first line and use the delete command to delete the current line. The vi editor saves the deleted line in buffer 1, and the screen and buffers look like Figure 6.2.

Figure 6.2
The Screen and Buffers After First Delete



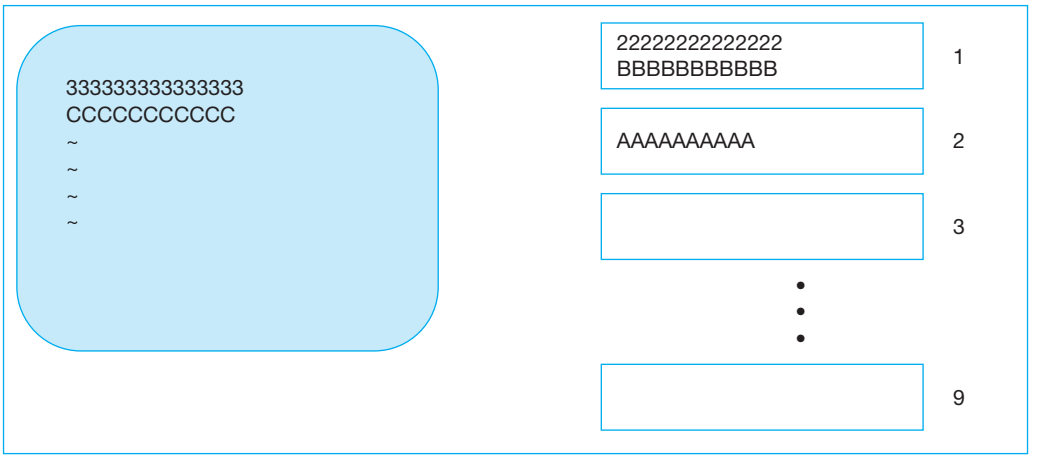
❑ Delete two lines using the delete command. The vi editor responds by deleting the two lines from your text and moving the contents of the temporary buffers one buffer up, to the empty buffer 1. Then it saves the deleted lines in buffer 1. The screen and buffers look like Figure 6.3.

1. *The two lines deleted are saved in one buffer. The numbered buffers are not storage for just a line but for any size of text you have changed. Any amount of text, whether it is one line or 100 lines, is saved in one buffer.*

2. *When all nine buffers are full, and vi needs buffer 1 for new material, the contents of buffer 9 are lost.*

Figure 6.3

The Screen and Buffers After Second Delete



❑ The portion of the text that you yank is also saved in the temporary buffers. Place the cursor on the first line and then type **yy**. vi responds by copying the line you yanked into buffer 1. Remember that vi has to move all the buffers' contents to the next buffers in order to empty buffer 1, and then the new material is copied into buffer 1. Refer to Figure 6.4.

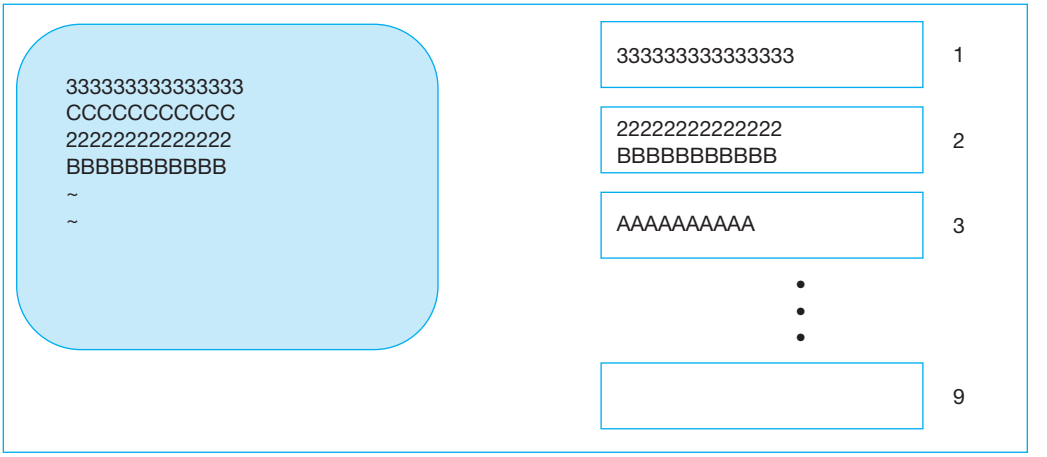
Figure 6.4

The Screen and Buffers After Yank

❑ Now, if you remember what you have in the numbered buffers, you can access any one of them by specifying its number as part of a command. For example, to copy the contents of buffer 2 to the end of the file, type **"2p**, and vi copies the contents of buffer 2 to right after the cursor position. Figure 6.5 shows the screen and contents of the buffers after this put command.

Accessing the buffers does not change their contents.

Figure 6.5
The Screen and Buffers After the Put Command



6.4.2 The Alphabetic Buffers

The vi editor also uses 26 named buffers. These buffers are named by the lowercase letters *a* through *z*, and you can refer to them by specifying their names explicitly. These buffers are similar to the numbered buffers, except the vi editor does not automatically change the contents of them every time you delete or yank from a file. It gives you more control over the operation. You can store deleted or copied text into a specified buffer, and later copy text from the named buffer to other places in your text by using the put operator. The text in a numbered buffer remains unchanged until you specify the buffer in a delete or yank operation. The format for specifying a buffer in your command is as follows:

```
double quotation mark + buffer name (a to z) + the command
```

To experiment with using commands to operate on a specific buffer, do the following things:

- Type **"wdd** to delete the current line and save a copy of it in buffer *w*.
- Type **"wp** to copy the contents of buffer *w* to the location indicated by the cursor position.
- Type **"z7yy** to copy seven lines into buffer *z*.
- Type **"zp** to copy the contents of buffer *z* (7 lines) to the location indicated by the cursor position.

1. **These commands are not displayed on the screen.**
2. **The alphabetic buffers are named in lowercase letters from *a* to *z*.**
3. **These commands do not require you to press [Return].**

6.5 THE CURSOR POSITIONING KEYS

The screen displays 24 lines of text at a time, and if your file contains more than 24 lines, you use the cursor movement keys to scroll a new line up or down into view. If your file contains 1,000 lines of text, you need about 999 keystrokes to bring line 999

into view on the screen. This is cumbersome and not practical. To overcome this problem, you use the vi editor's *paging operators.* Table 6.3 summarizes the paging operators (or command keys) and their capabilities.

Table 6.3
The vi's Paging Keys

Key	Operation
[Ctrl-d]	Scrolls the cursor down toward the end of your file, usually 12 lines at a time.
[Ctrl-u]	Scrolls the cursor up toward the beginning of your file, usually 12 lines at a time.
[Ctrl-f]	Scrolls the cursor down (forward) toward the end of your file, usually 24 lines at a time.
[Ctrl-b]	Scrolls the cursor up (backward) toward the beginning of your file, usually 24 lines at a time.

[Ctrl-d] means simultaneously holding down the [Ctrl] and [d] keys; this convention also applies to the other control keys.

If you have a really large file, even the scrolling commands are not practical to position the cursor. Another way to position the cursor is to use **G** prefixed with the line number on which you want to place the cursor.

To make line number 1000 the current line, do the following:

- Type **1000G** to move the cursor to line 1000.
- Type **1G** to move the cursor to the first line.
- Type **G** to move the cursor to the end of the file.

Another useful command is **[Ctrl-g]**, which tells you the line number of the current line. For example, if you press **[Ctrl-g]** while in the command mode, the vi editor responds by showing a message similar to the following:

```
"myfirst" line 30 of 90 – 30%
```

6.6 CUSTOMIZING THE vi EDITOR

The vi editor has many parameters (also called *options* or *flags*) that you can set, enable, or disable to control your working environment. These parameters have default values but are adjustable, and they include things like the tab setting, the right margin setting, and so on.

In order to see the complete list of parameters on the screen and how they are currently set in your system, enter the command mode, type **:set all**, and press **[Return]**.

Your terminal screen shows options similar to Figure 6.6. Your system may have other options set.

Figure 6.6
Screen Showing vi's Options

```
~
~
~
:set all                noshowmode
noautoindent            number          terms5wyse50
nobeautify              readonly        noterse
hardtabs=8              report=10       window=23
noignorecase            shiftwidth=8    wrapmargin=0
magic
[Hit return to continue]
```

6.6.1 The Options Formats

The set command is used to set the options, and the options fall into three categories, each set in a different manner:

- Boolean (toggle)
- Numeric
- String

Assuming there is an option called X, the following examples show how to set up the three categories of options.

The Boolean Options

The *Boolean options* work like a toggle switch: you can turn them on or off. These options are set by typing the option name and are disabled by adding the word *no* in front of the option name. Typing **set X** enables option X, and typing **set noX** disables option X.

There is no space between the word *no* and the option's name.

The Numeric Options

The *numeric options* accept a numeric value, and depending on the option, the range of the numeric value is different. Typing **set X=12** assigns value 12 to option X.

The String Options

The *string options* are similar to the numeric options, but they accept a string value. Typing **set X=PP** assigns string PP to option X.

There is no space on either side of the equal sign.

The Set Command

The **set** command is used to set the different vi environment options, list them, or get the value of a specified option. The basic formats of the **set** command are as follows; each command is completed by pressing **[Return]**:

```
:set all
```

shows all the options on the screen.

```
:set
```

shows only the changed options.

```
:set X?
```

shows the value of the option X.

6.6.2 Setting the vi Environment

The behavior of the vi editor can be customized by setting the edit parameters to new values, and there are different methods you can use to set them. The direct method of changing these values is to use the vi **set** command to set the values as desired. In this case, vi must be in the command mode before you can issue a **set** command. You can set every option using this method; however, the changes are temporary, and they are in effect only for your current editing session. When you quit the vi editor, your options settings are abandoned.

This section describes some useful vi parameters (listed alphabetically), and Table 6.4 summarizes these options. Most of the option names have abbreviations; you can use the full or abbreviated names in the set command.

Table 6.4
Some of vi's Environment Options

Option	Abbreviation	Operation
autoindent	**ai**	Aligns the new lines with the beginning of the previous ones.
ignorecase	**ic**	Ignores the uppercase/lowercase difference in search options.
magic		Allow use of the special characters in search.
number	**nu**	Displays line numbers.
report		Informs you of the number of lines affected by your last command.
scroll		Sets number of lines to scroll when **[Ctrl-d]** command is given.
shiftwidth	**sw**	Sets number of spaces to indent. Used with autoindent option.
showmode	**smd**	Displays the vi editor modes in the right corner of the screen.
terse		Shortens the error messages.
wrapmargin	**wm**	Sets the right margin to a specified number of characters.

***autoindent* Option** The *autoindent* (**ai**) *option* aligns each new line you type in the text mode with the beginning of the previous line. This option is useful for writing computer

programs in C, Ada, or other structured programming languages. You use **[Ctrl-d]** to backspace one level of indentation. While in text inset mode, each **[Ctrl-d]** backs up the number of columns specified by the *shiftwidth option*, discussed shortly. The default value for this option is set to **noai**.

***ignorecase* Option** The vi editor performs case-sensitive searches—that is, it differentiates between uppercase and lowercase letters. In order to make the vi editor ignore the letter cases, type **:set ignorecase** and press **[Return]**.

To restore vi to the case-sensitive search, type **:set noignorecase** and press **[Return]**.

***magic* Option** Certain characters (like bracket pairs **[ ]**) have a special meaning when you use them in search strings. When you toggle this option to **nomagic**, these characters no longer have special meanings. The default for this option is **magic**.

To set the *magic option*, you type **:set magic [Return]**, and to unset this option you type **:set nomagic [Return]**.

***number* Option** The vi editor does not ordinarily display the line numbers associated with each line. There are occasions when you want to refer to a line by its line number, and sometimes having the line numbers on the screen gives you a better feel for the size of a file and what part of the file you are editing.

To display the line numbers, type **:set number** and press **[Return]**.

The following screen shows the `myfirst` file after the *number option* has been set.

```
1 The vi history
2 The vi editor is an interactive text editor that is
supported
3 by most of the UNIX operating systems.
~
~
:set number
```

If you decide you do not want the line numbers to be displayed, type **:set nonumber** and press **[Return]**.

The line numbers are not part of the file; they appear on the screen only while you are using the vi editor.

***report* Option** The vi editor does not give you any feedback on your editing job. For example, if you type **5dd**, vi deletes five lines starting from the current line but does not show any confirmation message on the screen. If you want to see feedback related to your editing, use the **report** parameter of the set command. This parameter is set to the number of lines that must be changed before the vi editor displays a report of the number of lines affected.

To set the *report option* to affect two-line edits, type **:set report=2** and press **[Return]**. Then if your editing job affects more than two lines, vi displays a report on the status line. For example, deleting two lines (**2dd**) and copying two lines (**2yy**) produces the following reports on the bottom line, respectively:

```
2 lines deleted
2 lines yanked
```

If you want to receive feedback on every change to your file, you type **:set report=0 [Return]**. Now you will receive feedback even if one character is changed in your file.

***scroll* Option** The *scroll option* is set to the number of lines that you want the screen to scroll when using **[Ctrl-d]** (in command mode). For example, to have the screen scroll five lines, type **:set scroll=5** and press **[Return]**.

***shiftwidth* Option** The *shiftwidth* (**sw**) *option* sets the number of spaces used by the **[Ctrl-d]** key (in text input mode) when the autoindent option is in effect. The default setting for this option is **sw=8**. To change the setting to 10, for example, type **:set sw=10** and press **[Return]**.

***showmode* Option** The vi editor does not display any visual feedback to indicate whether it is in text input mode or command mode. This can be confusing, especially for beginners. You can set the *showmode option* to provide visual feedback on the screen.

To toggle on the showmode option, type **:set showmode** and press **[Return]**. Then, depending on which key you use to change from command mode to text input mode, vi displays a different message at the lower right side of the screen. If you press **A** or **a** to change mode, vi displays `APPEND MODE`; if you press **I** or **i**, vi shows `INSERT MODE`; if you press **O** or **o**, vi displays `OPEN MODE`, and so on.

These messages remain on the screen until you press **[Esc]** to change to command mode. When there is no message on the screen, vi is in the command mode.

To turn off the showmode option, type **:set noshowmode** and press **[Return]**.

***terse* Option** The *terse option* makes the vi editor display shorter error messages. The default for this option is **noterse**.

6.6.3 Line Length and Wraparound

Your terminal screen usually has 80 columns. When you type text and reach the end of the line (pass the 80th column), the screen starts a new line; that is what is called *wraparound.* The screen also starts a new line when you press **[Return]**. Thus, the length of a line on the screen could be any length from 1 to 80 characters. However, the vi editor starts a new line in your file only when you press **[Return]**. If you type 120 characters before pressing **[Return]**, your text appears in two lines on the screen, but in your file it is one line of 120 characters.

Long lines can be a problem when you print a file, and it is confusing to relate the number of lines on the screen to the actual number of lines in the file. The simplest way to limit the length of a line is by pressing **[Return]** any time before reaching the end of the line on the screen. Another way to limit the line length is to set the wrapmargin parameter and let the vi editor insert returns automatically.

***wrapmargin* Option** The *wrapmargin option* causes the vi editor to break the text you are entering when it reaches a specified number of characters from the right margin. To set the wrapmargin to 10 (where 10 is the number of the characters from the right side of the screen), type **:set wrapmargin=10** and press **[Return]**. Then, when what you are typing reaches column 70 (80 minus 10), the vi editor starts a new line, just as if you had pressed **[Return]**. If you are typing a word as the characters pass column 70, vi moves

the whole word to the next line. This means the right margin will probably be uneven. But remember, the vi editor is not a text formatter or a word processor.

The default value for the wrapmargin option is 0 (zero). To turn wrapmargin off, type **:set wrapmargin=0** and press **[Return]**.

6.6.4 Abbreviations and Macros

The vi editor provides you with some shortcuts to make your typing faster and simpler: **:ab** and **:map** are two commands that serve this purpose.

The Abbreviation Operator The **ab** (for *abbreviation*) command lets you assign short, abbreviated words that will automatically insert any string of characters. This helps speed up your typing. Pick an easy-to-remember abbreviation for text that you often type, and, after you have set up that abbreviation in the vi editor, you can use the abbreviated word instead of typing the entire text. For example, to abbreviate the words *UNIX Operating System,* which are used often in this text, to the abbreviation *uno,* type **:ab uno UNIX Operating System** and press **[Return]**.

In this example, *uno* is the abbreviation assigned to *UNIX Operating System*; thus, when vi is in text input mode, any time you type **uno** and then a space, vi expands *uno* to *UNIX Operating System.* If *uno* is part of another word, such as *unofficial*, no expansion occurs. It is the space before as well as after the *uno* that makes the vi editor recognize *uno* as an abbreviation and expand it.

To remove an abbreviation, you use the **unab** (for *unabbreviate*) operator. For example, to remove the *uno* abbreviation, type **:unab uno** and press **[Return]**.

To list the abbreviations that are set, type **:ab** and press **[Return]**.

1. *The abbreviations are assigned in the vi editor command mode and are used while you are typing in the text input mode.*
2. *The abbreviations set up are temporary; they remain in effect only during the current editing session.*

Try setting up some abbreviations as follows:

- Type **:ab ex extraordinary adventure** and press **[Return]** to assign *ex* to the string *extraordinary adventure.*
- Type **:ab 123 one, two, three, etc.** and press **[Return]** to assign *123* to the string *one, two, three, etc.*
- Type **:ab** and press **[Return]** to display all the abbreviations:

```
ex   extraordinary adventure
123  one, two, three, etc.
```

- Type **:unab 123** and press **[Return]** to remove the *123* abbreviation.

The Macro Operator The macro operator (**map**) lets you assign key sequences to a single key. Just as the abbreviation operator allows you to create a shortcut in text input mode, so **map** lets you create shortcuts in command mode. For example, to assign the command **5dd** (delete five lines) to **q**, type **:map q 5dd** and press **[Return]**. Then, while vi is in command mode, vi deletes five lines of text each time you press **q**.

To remove a **map** assignment, you use the **unmap** operator. Type **:unmap q** and press **[Return]**.

To look at the list of the map keys and their assignments, type **:map** and press **[Return]**.

The vi editor uses most of the keys on the keyboard for commands. This leaves you with a limited number of keys to assign to your key sequences. Available keys are **K**, **q**, **V**, **[Ctrl-e]**, and **[Ctrl-x]**.

You can also assign the function keys of your terminal with the **map** command. In this case, you type **#n** as the key name, where *n* refers to the function key number. For example, to assign **5dd** to **[F2]**, type **:map #2 5dd** and press **[Return]**.

Then, if you press **[F2]** while in command mode, vi deletes five lines of text.

The following examples show some key assignments:

- Type **:map V /unix** and press **[Return]** to assign **[V]** to the search command, searching for *unix*.
- Type **:map #3 yy** and press **[Return]** to assign **[F3]** to yank a line.
- Type **:map** and press **[Return]** to display the map key assignments.

```
V   /unix
#3  yy
```

Suppose you want to find the word *unix* in your file and replace it with *UNIX*. Follow this key sequence (search and replace commands are explained later in this chapter):

- Type **:/unix** and press **[Return]** to search for the word unix.
- Type **cwUNIX** and press **[Esc]** to change the word *unix* to *UNIX* and return vi to command mode.

In map key assignment, you press **[Ctrl-v][Return]** to represent **[Return]**, and **[Ctrl-v][Esc]** to represent the **[Esc]** key in the command line. Thus, to map the preceding command sequences to a single key, say **V**, type **:map V /unix** and press **[Ctrl-V][Return]** and type **cwUNIX** and press **[Ctrl-V][Esc]**. This command line uses unprintable characters (**[Ctrl-V]** and **[Esc]**), so what you see on the screen is the following:

```
:map V /unix ^McwUNIX ^[
```

1. *The map keys you create in the vi editor are temporary; they are in effect only for the current editing session.*
2. *The map keys are assigned and used while vi is in the command mode.*

You must precede [Return] and [Esc] with [Ctrl-v] if they are part of a map key assignment.

6.6.5 The `.exrc` File

All the options that you set up while you are in the vi editor are temporary; they disappear when you exit vi. To make the options permanent and save yourself the trouble of retyping them for every editing job, you can save the options settings in a file called `.exrc`.

Files starting with a . (dot) are called hidden files; you learned about these in Chapter 5.

When you start the vi editor, it automatically checks for the existence of the `.exrc` file in your current (working) directory and sets up the edit environment according to

what it finds in the file. If vi does not find the .exrc file in the current directory, it checks your HOME directory and sets up options according to the .exrc file it finds there. If vi does not find a .exrc file, it assumes the default values of the options.

The way vi checks for the existence of the .exrc file gives you a powerful tool to create specialized .exrc files for your different editing needs. For example, you may create a general-purpose .exrc file under your HOME directory and a different .exrc file for your C programs under the directory in which you keep your C programs. You can use the vi editor to create a .exrc file or to modify it, if it already exists.

To create an .exrc file, type **vi .exrc** and press **[Return]**. Then enter the set and other commands you want to use (perhaps like those on the following screen).

```
set report=0
set showmode
set nu
set wm=10
ab uop UNIX Operating System
map q 5dd
```

Do not forget the . (dot) at the beginning of the filename.

If you create this file under your HOME directory, the edit environment will be established each time you use vi. Under the settings shown, vi displays what mode it is in, shows the line number, sets the right margin at 10 characters (line length is 70), inserts *UNIX Operating System* into your file every time you type **uop** followed by a space, and deletes five lines each time you press **q**.

1. .exrc *belongs to a group of files called startup files.*
2. *There are other utilities that use startup files similar to the* .exrc *file.*

6.7 THE LAST OF THE GREAT vi COMMANDS

Before concluding this discussion of the vi editor, we must consider one more vi operator. You know enough vi editor commands and other particulars so that you can easily and efficiently create or modify files. However, that is not all that vi can do. The vi editor has more than 100 commands and numerous variations of them that, when combined with the scope of the commands, give you detailed control over your editing job.

6.7.1 Running Shell Commands

You can run UNIX shell commands from the vi command line. This handy feature lets you temporarily put the vi editor aside and go to the shell commands. The **!** (exclamation mark) signals vi that the next command is a UNIX shell command. For example, to run the date command while in the vi editor, type **:! date** and press **[Return]**. The vi editor

clears the screen, executes the command **date**, and you see lines similar to the following on the screen:

```
Sat Nov 27 14:00:52 EDT 2005
[Hit any key to continue]
```

Pressing a key returns the vi editor to the screen, and you can continue editing where you left off. If you want, you can also read the result of the shell commands and add it to your text. You use the **:r** (read) command followed by **!** to incorporate the command's result into your editing file.

To read the system's date and time, type **:r ! date** and press **[Return]**; vi responds by putting the current date and time under the current line.

The vi editor remains in the command mode.

```
The vi history
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history
The vi editor is an interactive text editor which is
Sat Nov 27 14:00:52 EDT 2005
supported by most of the UNIX operating systems.
```

The following command sequences show the use of **!**:

- Type **:! ls** and press **[Return]** to list the files in the current directory.
- Type **:! who** and press **[Return]** to show who is on the system.
- Type **:! date** and press **[Return]** to show the date and time of day.
- Type **:! pwd** and press **[Return]** to list the contents of the working directory.
- Type **:r ! date** and press **[Return]** to read the results of the **date** command and place it after the cursor position.
- Type **:r ! cal 1 2005** and press **[Return]** to read the calendar for January 2005 and place it after the cursor position.
- Type **:! vi mylast** and press **[Return]** to invoke another copy of vi to edit the `mylast` file.

6.7.2 Joining Lines

Use **J** to join two lines together. The **J** command joins the line below the current line to the current line, right after the cursor position. If the joining of the two lines results in a long line, vi wraps it around the screen.

To join two lines together, do the following:

- Use the cursor movement keys to place the cursor at the end of the first line.
- Press **J**. vi joins the line below the current line to the current line.

```
The vi history_
The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

```
The vi history The vi editor is an interactive text editor that is
supported by most of the UNIX operating systems.
```

6.7.3 Searching and Replacing

There are occasions when you want to change a word throughout a file. If the file is a long one, it is cumbersome to go through the text, find each occurrence of a specific word, and change it. Additionally, the chances are good that you will miss one or two occurrences of the word. A better way is to use the vi search commands (**/** and **?**) in combination with other commands to do the job.

The following command sequence demonstrates the vi search and replacement capability:

- Type **:/UNIX** and press **[Return]** to search forward to find the first occurrence of the word *UNIX*.
- Type **cwunix** and press **[Return]** to change *UNIX* to *unix*.
- Type **n** to find the next occurrence of the word *UNIX*.
- Press **.** (dot) to repeat the last change (*UNIX* to *unix*.)
- Type **:?unix** and press **[Return]** to search backward from the current line to find the first occurrence of the word *unix*.
- Type **dw** to delete the word *unix*.
- Type **n** to find the next occurrence of the word *unix*.
- Press **.** (dot) to repeat the last command (**dw**) and delete the word *unix*.

6.7.4 File Recovery Option

What if the computer or vi crashes while you are editing a file? Fortunately, vi provides a file recovery option that recovers the version of file that you were editing (it was in the buffer) when the system or vi crash occurred. In most cases recovery is quite easy. You start vi with the -r option on the same file you were editing when the crash happened.

For example, the following command line recovers `myfirst` file:

```
$ vi -r myfirst [Return]    . .  Invokes vi recovery option.
```

If you were editing without a filename, or you don't remember the name of the file you were editing, just type the vi -r command without the filename argument.

```
$ vi -r [Return]    . . . . . .  Invokes vi recovery without a filename.
```

In this case, vi displays a list of the saved files, similar to the list in Figure 6.7.

Figure 6.7
Using the vi File Recovery Option

```
$ vi -r
/usr/preserve/david:
On Wed Aug 22 at 15:23, saved 2 lines of file "myfirst"
On Mon Oct 06 at 09:12, saved 6 lines of file "yourfirst"
/var/tmp:
No files saved.
$
```

Linux vim Editor

As was mentioned in Chapter 4, Linux provides the vim editor that is upwardly compatible to the vi editor. In fact, every time you use vi, the vim editor is invoked. Like vi, vim can be used to edit any text file. However, vim provides a lot of enhancements beyond the capablities of vi, including sophisticated file recovery and the help commands.

***help* Command** While in the vi editor, you type the following command:

```
:help recovery [Return]    . . .  Display the file recovery usage.
```

A full description of the file recovery commands will be displayed.

***recover* Command** When you start vi to edit a file you might get the `"ATTENTION: Found a swap file ..."` message. This means something happened the last time you were editing this file and a copy of the file was saved before the crash. In this case, you use the recover command to recover your file.

```
:recover [Return]    . . . . . .  Recover the current file.
```

Use the **man** command to obtain more information about the vim editor and its capabilities.

COMMAND SUMMARY

The following vi editor commands and operators discussed in this chapter. These commands complement the commands you learned in Chapter 4.

Cut-and-paste keys

These keys are used to rearrange text in your file.
They are applicable in vi's command mode.

Key	Operation
d	Deletes a specified portion of the text and stores it in a temporary buffer; this buffer can be accessed by using the put operator.
y	Copies a specified portion of the text into a temporary buffer; this buffer can be accessed by using the put operator.
P	Places the contents of a specified buffer above the cursor position.
p	Places the contents of a specified buffer after the cursor position.

Paging keys

The paging keys are used to scroll a larger portion of your file.

Key	Operation
[Ctrl-d]	Scrolls the cursor down toward the end of the file, usually 12 lines at a time.
[Ctrl-u]	Scrolls the cursor up toward the beginning of the file, usually 12 lines at a time.
[Ctrl-f]	Scrolls the cursor down (forward) toward the end of the file, usually 24 lines at a time.
[Ctrl-b]	Scrolls the cursor up (backward) toward the beginning of the file, usually 24 lines at a time.

Scope keys

Using the vi commands in combination with the scope keys gives you more control in your editing tasks.

Key	Operation
$	The scope is from the cursor position to the end of the current line.
0 (zero)	The scope is from just before the cursor position to the beginning of the current line.
e or **w**	The scope is from the cursor position to the end of the current word.
b	The scope is from the letter before the cursor backward to the beginning of the current word.

Setting the vi environment

You can customize the behavior of the vi editor by setting the vi environment options. You use the **set** command to change the options' values.

Option	Abbreviation	Operation
autoindent	**ai**	Aligns the new lines with the beginning of the previous ones.
ignorecase	**ic**	Ignores the uppercase/lowercase difference in search operations.
magic		Allows the use of the special characters in a search.
number	**nu**	Displays line numbers.
report		Informs you of the number of lines affected by the last command.
scroll		Sets the number of lines to scroll when **[Ctrl-d]** command is given.
shiftwidth	**sw**	Sets the number of spaces to indent; used with the autoindent option.
showmode	**smd**	Displays the vi editor modes in the right corner of the screen.
terse		Shortens the error messages.
wrapmargin	**wm**	Sets the right margin to a specified number of characters.

REVIEW EXERCISES

1. What is the command line to open a file named `xyz` in read only mode?
2. What is the command line to open a file named `xyz` with cursor on the line with first occurrence of word UNIX?
3. What is **view**?
4. What are the numbered buffers?
5. What are the named buffers?
6. What is the vi command to
   a. delete a line?
   b. delete a word?
   c. copy a line?
   d. copy a word?
   e. delete to the end of the current line?

f. save two lines in a buffer called *z*?

g. copy the contents of the *z* buffer after the current line?

h. copy the contents of the buffer two after the current line?

7. What is the vi command for removing the showmode option?

8. What is the vi command to abbreviate "one two three" to "123"?

9. What is the .`exrc` file? When is the .`exrc` file is executed?

10. What is the command to execute a shell command such as **date** while in vi?

11. What is the vi command to read the current date and place it in the file under the current line?

12. What is the **set** command for vi to display confirmation messages?

13. What is the vi file recovery option?

14. What is the command to get the list of the file that were saved before a crash?

15. What is the vim editor?

16. Match the commands shown in the left column with explanations shown in the right. All the commands are applicable only in the command mode:

1.	**G**	a.	Replaces the character under the cursor with the letter *x*.
2.	**/most**	b.	Places the cursor on the last line in the file.
3.	**[Ctrl-g]**	c.	Copies four lines in buffer *x*.
4.	**2dw**	d.	Moves cursor down one line.
5.	**j**	e.	Shows the line number of the current line.
6.	**"x4yy**	f.	Positions the cursor on line 66.
7.	**$**	g.	Deletes the character under the cursor.
8.	**0 (zero)**	h.	Retrieves the contents of buffer 1.
9.	**66G**	i.	Deletes two words.
10.	**x**	j.	Positions the cursor at the end of the current line.
11.	**rx**	k.	Finds the word *most*.
12.	**"lp**	l.	Positions the cursor at the beginning of the current line.

17. While using the vi editor, what command do you use to

a. set the line number option?

b. save five lines in buffer *x*?

c. read (import) the date string into your file?

d. list your current directory?

e. create an abbreviation?

f. remove an abbreviation?

g. read another file?

h. write (save) a file without quitting the vi editor?

i. delete a word?

Terminal Session

In this terminal session, create a file called `garden` and practice using the editing keys discussed in this chapter. Create this file as shown in Screen 1. Then use cut-and-paste, cursor positioning, and other commands to make it look like Screen 2. Finally, apply the following commands to your `garden` file.

1. Create an abbreviation of your name and add it to the beginning of your file.
2. Create a map key that finds a line with a specific word and deletes that line.
3. Undo the previous text changes.
4. While in vi, list your file in your current directory.
5. Read the date and time of the day, and place it after your name in the `garden` file.
6. Read another file (import it), and add it to the end of the `garden` file.
7. Save your file with another name.
8. Set the showmode option.
9. Set the line number option.
10. Move the cursor to the end of the file.
11. Move the cursor to the top of the file.
12. Move the cursor to line 10.
13. Search for the word *weeds*.
14. Set the report option to 1.
15. Remove the line number option.
16. Delete five lines; observe the vi feedback message. Use **U** to undo your deletion.
17. Copy five lines from beginning of the file to the end of the file. Observe the vi feedback message. Use **U** to undo your last editing.
18. Use the paging keys, **[Ctrl-d]**, **[Ctrl-u]**, and so on, and observe the results.
19. While in vi, copy five lines of your current file and save it in another file.
20. Add the current date at the beginning of your file using the **date** command.

Screen 1

```
Everywhere the trend is toward a simpler, and easy to care
garden. Few advises might help you to have less trouble with
your gardening. I am sure you have heard them before, but
listen once more. Gardening: The easy approach visit the plant
nurseries, it is good for your soul. Let me tell you that
There is no easy to care garden. Use plants that are suitable
for your climate. Native plants are good CHOICE. Before
planting, choose the right site. Use your imagination, plants
grow faster than what you think. Gardening can be made easier
and more enjoyable if you hire a gardener to do the job. Use
mulches to reduce weeds and save time in watering the plants.
Do not use too much chemicals to kill every weed insight. You
are the only one who sees the weeds, let them grow. They keep
the moisture and prevent soil erosion.
```

Screen 2

```
Gardening: The easy approach
Everywhere the trend is toward making simpler and easier-to-
care-for gardens.
However, let me tell you that there are no easy-to-care-for
gardens.
Gardening can be made easier and more enjoyable if you hire a
gardener to do the job.
Some advice might help you to have less trouble with your
gardening.
I am sure you have heard it before, but listen once more:
1. Before planting, choose the right site.
   Use your imagination. Plants grow faster than you think.
2. Visit the plant nurseries; it is good for your soul.
3. Use mulch to reduce weeds and save time in watering.
4. Use plants that are suitable for your climate.
   Native plants are a good choice.
5. Do not use too many chemicals to kill every weed in sight.
   You are probably the only one who sees the weeds.
   Let them grow.
   They keep moisture and prevent soil erosion.
```